International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Performance Enhancement of MapReduce Framework in Big Data Application Using Load Balancing with Cache

Sushant Shirish Nagavkar[1], Ashishkumar[2]

[1]M.E. Computer Network Student from G.H. Raisoni Collage of Engineering and management, Ahemadnagar, India

[2]Assistant Professor of G.H. Raisoni Collage of Engineering and management, Ahemadnagar, India

Abstract: ***Hadoop is open source software that is used to store big data, it supports data demanding applications and performs analysis, using a random placement method for parallel processing to give effortlessness and load balance. To achieve maximum parallelism per group to load balance a new Data-gRouping-AWare (DRAW) data placement is used. Problem in big data is when any query executes repeatedly it repeats whole process of execution to obtain result. In MapReduce framework and generates a large amount of intermediate data. Such huge amount of information is thrown away after the tasks finish, because MapReduce is not able to use this data. Dache, a data-aware cache framework for big-data applications gives the produced intermediate results to the cache manager. Task inquiries the cache manager before performing the actual computing work.***

Keywords: BEA, Big-data, caching, DACH, DRAW, Hadoop, HDFS, MapReduce

1. Introduction

Big data belongs to datasets whose size is outside the capacity of usual database software tools to capture, store, manage, and analyze [2]. “Big Data", described by the unusual volume of data, data generation velocity, and structural variety of data, support for extensive data analytics form a mainly challenging task [5]. The main aim of big data is to help companies make better business decisions by facilitating data scientists and users to analyze huge volumes of transaction data and other data sources [6]. With the help of predictive analytics and knowledge mining big data can be easily processed but due to unstructured data it may not fit in traditional data warehouse [6]. But traditional data warehouses are unable to handle the processing demands of big data. Apache is founded by Hadoop and it is a software framework for processing large datasets. It uses Hadoop Distributed File System (HDFS) for storage purpose and MapReduce for processing components of Hadoop [2].

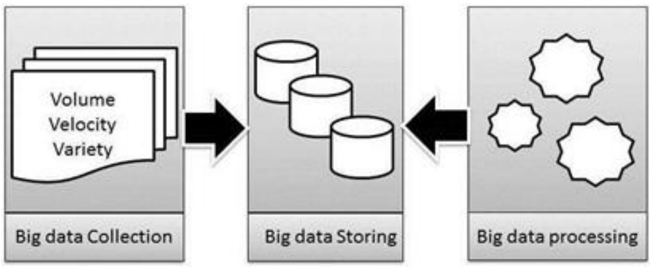

Figure 1: General Architecture of big data analytics.

For the large-scale processing and analysis of vast data sets MapReduce is the most popular framework. MapReduce programming is useful for processing large datasets. MapReduce uses 2 functions: Map and Reduce function. User can write the Map function which takes input and produces a set of key/value pairs. This all produced values with the same intermediate key I is grouped by the MapReduce library and then passes it to the Reduce function. The Reduce function is also written by the user which accepts and a set of intermediate key I and values for that key. It merges these values to obtain probably smaller set of values [2].

The main function of MapReduce framework executes on a single master machine where input data is preprocessed before map functions are called and/or post process the output of reduce functions. As per need of applications, a pair of map and reduce functions may be executed once or more time. The research area has newly received a lot of attentions for developing MapReduce algorithms for examine big data [7].

1.1 Big Data

Big data refers to datasets whose size is away from the capacity of typical database software tools to capture, store, manage, and analyze. The possible sources of big data are:

Traditional project data contains customer information from CRM systems, Transactional ERP data, Web store transactions, and common ledger data. Data like Call Detail Records (CDR), weblogs, smart meters, manufacturing sensors, equipment logs, and trading systems data are machine generated data. A customer feedback stream, micro-blogging sites like Twitter and social media platforms like facebook are comes under social data. There are some other sources of data like Health care, Public sector, Retail and Manufacturing [2].

1.1.1 Characteristics of Big data

Big data is a term used to describe the collection of large and complex data sets that are difficult to process using on hand database management tools or traditional data processing applications. Big data spans across seven dimensions which include volume, variety, volume, value, veracity, volatility and complexity [4].

Volume: The volume of data here is very huge and is generated from a lot of different devices. The size of the data is usually in terabytes and petabytes. All this data also needs to be encrypted for privacy protection.

Velocity: This describes the real time attribute found in some of the data sets for example streaming data. The result that misses the appropriate time is usually of little value.

Variety: Big data consists of a variety of different types of data i.e. structured, unstructured and semi-structured data. The data maybe in the form of blogs, videos, pictures, audio files, location information etc.

Value: This refers to the complex, advanced, predictive, business analysis and insights associated with the large data sets.

Veracity: This deals with uncertain or imprecise data. It refers to the noise, biases and abnormality in data. This is where we find out if the data that is being stored and mined is meaningful to the problem being analyzed.

Volatility: Big Data volatility refers to how long the data is going to be valid and how long it should be stored.

Complexity: A complex dynamic relationship often exists in Big data. The change of one data might result in the change of more than one set of data triggering a rippling effect.

1.2 Mapreduce

For processing and generating large dataset MapReduce programming model is used. Users can write a map function that processes a key/value pair to produce a set of intermediate key/value pairs, and a reduce function is capable of merging this all intermediate values associated with the same intermediate key.

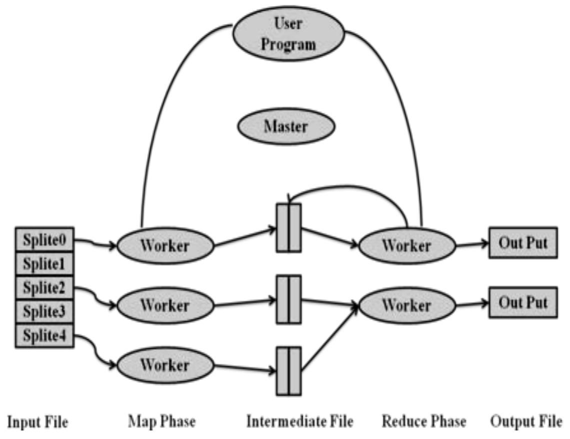

Figure 2: MapReduce programming model [1].

MapReduce is popular due to its simple programming interface and excellent performance while applying a large range of applications. Such applications receive a huge amount of input data and also called as “Big-data applications”. As shown in Fig. 2, input data is first divide and then supply to workers in the map phase. Individual data substances are called records. Each worker got this splinted input from MapReduce system to produces records. Intermediate results generated in the map phase are shuffled and sorted by the MapReduce system and are then give to the workers in the reduce phase. Final results are produced by number of reducers and written to the disk.

Conceptually the map and reduce functions gave by the user have connected Types:

Map (k1, v1) → list (K2, v2)
Reduce (K2, list (v2)) → list (v2)

I.e., the input keys and values are drawn from a different domain than the output keys and values. Furthermore, the intermediate keys and values are from the same domain as the output keys and values [9].

This paper is organized as follows: Section II describes literature survey, which explains the work done earlier by different authors. Section III explains the existing system for big data processing for parallel processing and load balance. Section IV explains about proposed system, its architecture, programming model and algorithm. Section V gives the idea about expected results of proposed system and Section VI briefs the conclusion.

2. Literature Survey

Several previous works exploited the grouping-like data semantics and organized data layout in some specific ways to support high-performance data accesses.

In [3] Qi Chen, Cheng Liu, and Zhen Xia, in MapReduce system straggler machines impact seriously straggler machines takes an unusually long time to finish tasks. System performance affect by: data skew, asynchronous task starts, improper configuration of phase pro-portion and unexpected resource competitions. To overcome these issues they develop a new strategy called maximum cost performance (MCP). Aim’s is not only reducing the job execution time but also throughput improving the cluster.

In [4] Shanjiang Tang, Bu-Sung Lee, Bingsheng He faces the problem of slot based MapReduce system can suffers from poor performance due to its un-optimized resource allocation. They find the performance degraded due to first Map and Reduce slots may empty because of pre-configuration; sec-ond system can face the straggler problem and third delay scheduling of MapReduce systems to overcome this they uses following techniques to improve performance as 1) empty slots are reallocated by using dynamic Hadoop slot allocation (DHSA); 2) straggler problem is solved by using speculative execution performance balance (SEPB); 3) delay scheduling is overcome with slot pre-scheduling.

In [10] X. Jiong, Y. Shu, R. Xiaojun, D. Zhiyang, T. Yun, J. Majors, A. Manzanares and Q. Xiao identified performance problem in Hadoop Distributed File System on heterogeneous clusters. Inspired due to performance degradation caused by heterogeneity, they have designed and implemented a data placement mechanism in witch input files are fragmented and provides to heterogeneous nodes based on their computing ca-pacities. Main approach is to improve performance of Hadoop heterogeneous clusters. Future research is focuses on control the data redundancy issue and designing a dynamic data dis-tribution mechanism for several data demanding applications working collectedly.

3. Existing System

Current years have seen an increasing amount of data parallel computing methods such as MapReduce and Hadoop to run data demanding applications and perform analysis. This data location scheme can suggestively improve the performance. Present data parallel frameworks distribute the data using a random allocation method for simplicity and load balance. A co-related data is probably processed as a group by specific domain applications. Here, we officially define the data grouping to characterize the probability of two or more data that can be accessed as a single group. Such data grouping can be measured by a weight: a number that these data have previously accessed as a group. The overall data distribution may be balanced using Hadoop default random data placement strategy; but there is no guarantee that the data accessed as a group is uniformly distributed. Due to uniformly distributed grouping data, some map tasks are either scheduled on different nodes which remotely access the required data, or scheduled on these data allocated nodes but have to wait in the queue.

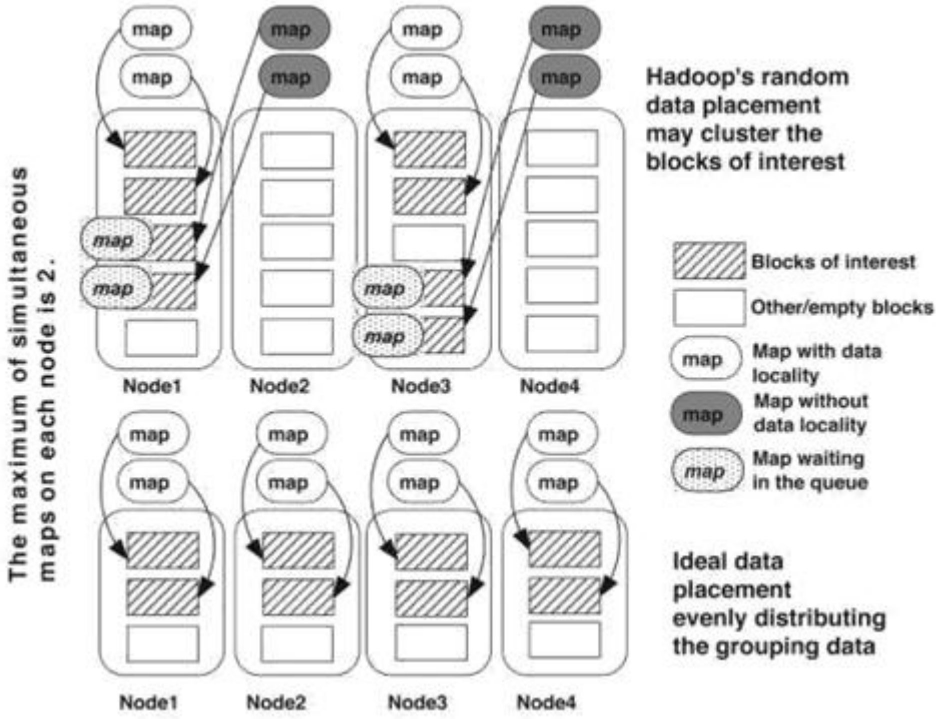
map
Node1
Node2
Node3
Node4
Hadoop's random data placement may cluster the blocks of interest
Blocks of interest
Other/empty blocks
Map with data locality
Map without data locality
Map waiting in the queue
Ideal data placement evenly distributing the grouping data
The maximum of simultaneous maps on each node is 2.

Figure 3. Simple case showing the efficiency of data placement for MapReduce programs[1]

In Fig.3 if the data groupings are distributed by Hadoops random strategy, the covered map tasks with either remote data access or queuing interval are the performance barriers; this barrier is avoided by evenly distribution of data, with the MapReduce program.

When examine the opportunity for random data distribution to evenly distribute the data from the same group. Surveillance shows this possibility is affected by three factors: 1) the number of replica for every data block in each rack (NR); 2) the maximum number of concurrent map tasks run on each node (NS); and 3) access patterns of the data groups. Hadoops default random solution will reach the ideal distribution: a) assume NR is very big, i.e. each node will hold one copy of the data therefore the maximized parallelism can be reached; b) assume NS is very big, i.e. all the map tasks can run concurrently hence the performance will not be degraded[1].

There are three portions in our scheme: a history data access graph (HDAG) to feat system log files to understand the data grouping information; a data grouping matrix (DGM) to count the grouping weights among the data and produce the improved data groupings; an optimal data placement algorithm (ODPA) to procedure the optimal data placement[1].

- **History Data Access Graph (HDAG):** HDAG is used to produce graph description on the basis of file access patterns, which can be obtained from the history of data accesses. In Hadoop each cluster rack, maintains system logs recording at Name Node for every operation of system, with the help of files which have been accessed. A simple solution is: observer the files which are accessed; every two continuously accessed files will be considered in the same group. This solution is simple for execution due to it only needs a traversal of the Name Node log files[1].

- **DGM:** On basis of HDAG, we can produce a DGM witch show the relation between every two data blocks. DGM is a n by n matrix, where n is the number of blocks present in system. As we mentioned earlier, same data may belongs to group A and group B at the same time; In the DGM grouping weight indicates how likely one data should be grouped with another data. The relationships among the data in DGM required making sure that blocks on the same node have minimal chance to be in the same group[1].

- **ODPA:** To achieve the optimal data placement only knowledge of data groups is not sufficient. It must require an algorithm named ODPA to complete our system. This algorithm is based on sub matrix for ODPA (OSM) from clustered data grouping matrix (CDGM). OSM shows the dependencies between the data previously placed and the ones being placed. With the help of ODPA, DRAW can succeed the two goals: maximize the parallel distribution of the grouping data, and balance the complete storage loads[1].

3.1 Algorithm 1: Bond Energy Algorithm

Table 1: Model notations

AFF	Attribute Affinity matrix
QA	Query Access matrix
CA	Clustered Affinity matrix
DM	Distance Matrix
AU	Attribute Usage matrix
TSC	Total Storage Cost
V	Volume of data allocation measured in characters
SCij	Storage cost of fragment i in site j
Aff(Ai;Aj)	The affinity of attributes Ai and Aj
Freql(qk)	Access frequency of a query k on site l
Accl(qk)	Access per execution of query k on site l
Sij	Similarity measure between Ai and Aj
MQA	Minimized Query Access
SC	Storage Cost
IIC	Iteration Input Cluster(is fed to next iteration)
LC	Leaf Cluster

```
Require:
    Attribute Query Matrix
    Query Access Matrix
Result:
        AFF Matrix
1: $S \leftarrow MQA$
2: for each attribute number $i$ do
3:  $QS_i \leftarrow sum(S_{ij})$
4: end for
5: for each attribute number $i$ do
6:    for each attribute number $j$ do
7:        initialize $n_{00}; n_{11}; n_{01}; n_{10}$ by $0$
8:    if $(i == j)$ then
9:        $AFF_{ij} \leftarrow sum(A_j) * QS$
10:           else
11:   for each query number $k$ do
12:       calculating $n_{00}; n_{11}; n_{01}; n_{10}$
13: if $(n_{01} == 0$ and $n_{10} > 0)$ or $(n_{10} == 0$ and $n_{01} > 0)$ then
14:       $coef \leftarrow (-1)(n_{01} + n_{10}) * w_1$
15:     else
16:       $coef \leftarrow (|n_{01} - n_{10}|) * w1$
17:   end if
18:        $S_{ij} (n_{11} + w_2 * n_{00}) / (n11 + w_2 * n_{00}) + coef$
19:          end for
20:       end if
21:    $AFF_{ij} \leftarrow S_i * QS_i$
22:   end for
23: end for
24: call Function Split(AFF)
```

3.2 Algorithm 2 : ODPA algorithm

Input: The sub-matrix (OSM): M[n][n];Where n is the number of data nodes;

Output: A matrix indicating the optimal data placement: DP[2][n];

```
Steps:
For each row from M[n][n] do
    R= the index of current, row;
    Find the minimum value V in this row; A set MinSet;
    MinSet = C1,V1,C2,V2; // there may be more than one
    minimum value
if there is only one tuple (C1,V1) in MinSet then
//The data referred by C1 should be placed with the data
referred by R on the same node;
    DP[0][R] = R;
    DP[1][R] = C1;
    Mark column C1 is invalid (already assigned);
Continue;
end if
    for each column Ci from MinSet do
    Calculate Sum[i] = sum(M[*][Ci]) ;// all the items in Ci
    column
end for
    Choose the largest value from Sum array;
    C= the index of the chosen Sum item;
    DP[0][R] = R;
    DP[1][R] = C;
    Mark column C is invalid (already assigned);
end for
```

3.3 Mathematical Model

BEA uses affinity of attributes to create clusters of attributes, which are the most similar. It starts with Attribute Usage (AU) and Query Access (QA) matrices generates Attribute Affinity matrix (AFF) and finally creates Clustered Affinity matrix (CA) by positioning and re-positioning columns and rows of attributes. The Affinity measure is too simple. The proposed Affinity measure in BEA is basically based on simultaneous access of attribute Ai and attribute Aj of relation R(A1;A2; :::;An) by query qk for every query in Q = (q1; q2; :::; qq): In other words, Two attributes are considered similar if they are accessed by the same query. This is indicated in AU by Aij = 1 and Aik = 1 simultaneously for attributes j and k accessed by query i Considering the Affinity of attributes Ai and Aj as aff(Ai;Aj), access frequency of a query k on site l as freql(qk), and access per execution of query k on site l as accl(qk); the equation for Affinity presented is as below.

$$aff(Ai;Aj) = \sum_{k|use(qk;Ai)=1 \wedge use(qk;Ai)=1} \sum_{\forall sl} freql(qk) * accl(qk)$$

In the process of splitting the bond between two attributes i and j and the net contribution to the global affinity measure of placing the attribute k between i and j play key roles. The bond between attributes i and j is defined as

$$bond(Ai;Aj) = \sum_{k=1}^{n} aff(Ak;Aj)\, aff(AkAj)$$

The net contribution of placing the attribute k between i and j is defined as
cont(Ai;Ak;Aj)=2bond(Ai;Ak)+2bond(Ak;Aj)-2bond(Ai;Aj)

The split function generates the Clustered A_nity Matrix in two steps.

4. Proposed System

4.1 Problem definition

Googles MapReduce and Apaches Hadoop, it is open-source implementation, are the defected software systems for big-data applications. A study of the MapReduce framework is that the framework produces a large amount of intermediate data. Such ridiculous information is thrown away after the tasks finish, because MapReduce is incapable to utilize them.

4.2 Execution plan

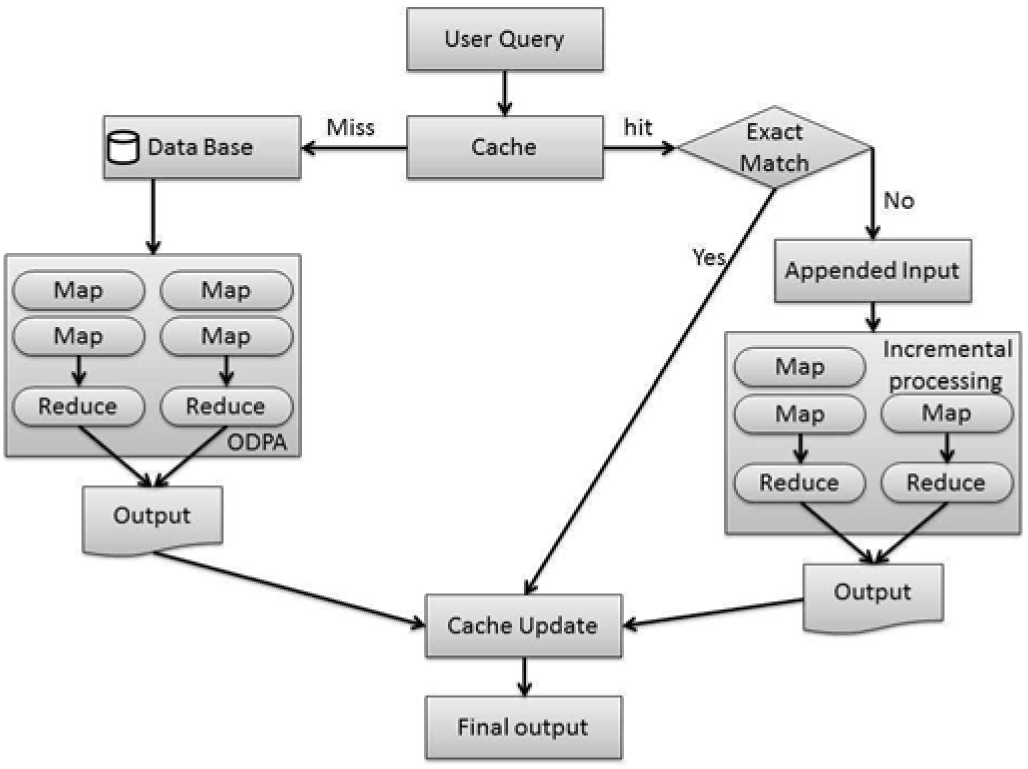

Figure 4: Proposed system architecture

Fig.4 shows the scheme that identifies the source input from which a cache item is achieved, and the operations applied on the input, so workers produces cache item which are indexed properly in map phase. We also have a method for reducers to apply the cached results in the map phase to increase speed of execution of the MapReduce job.

A worker node/process contacts the cache manager every time before it starts processing on given input data file. The worker process passes the file name and the operations that it going to perform on the file to the cache manager. The cache manager takes this message and compares it with the stored mapping data. If there is an exact match to a cache item, i.e., it is the same as the file name of the request and its operations, then the cache manager will send back a reply enclosing the tentative description of the cache item to the worker process.

The worker process accepts the tentative report and gets the cache item. For further processing, the worker sends the file to the next-stage worker processes.

5. Expected Results

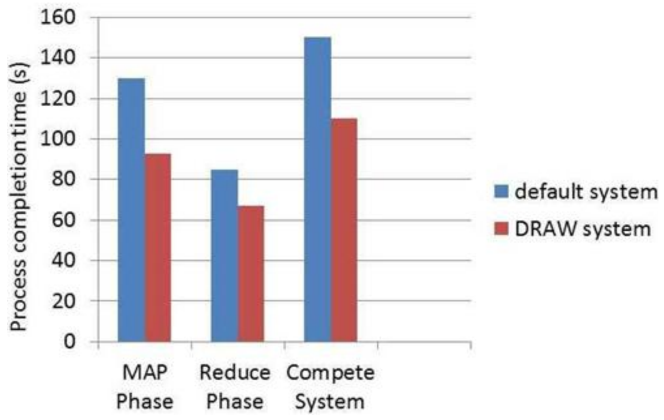

Figure 5: Execution time of the system

Fig. 5 shows that DRAWs regrouped data and Hadoops randomly placed data. The number of reducers are used so that the reduce phase will not produce bottleneck. DRAW finished map phase nearly about 30% earlier than the default placed data, and the tasks overall execution time is also 25% better by using DRAW.

In proposed system data is appended at the input file. The size of the appended data varies and is represented as a percentage number to the original input file size, which is in GBs. As a result Dache can avoid computation tasks that take extra time, which achieves more speedups. Dache is able to complete jobs 20% faster than Hadoop in all situations. It shows that proposed system takes less time for processing as compare to existing system.

In proposed system CPU utilization ratio of program is calculated by averaging the CPU utilization ratio of MapReduce job processing time. Hadoop 30% takes more CPU cycles than Dache, which is expected by the CPU-bound nature of the execution procedure. It is clear that Dache saves a major amount of CPU cycles, which is proved by the much lower CPU utilization ratio.

6. Conclusion

The proposed system in this paper uses the Hadoop implementation of MapReduce which is responsible for parallel processing on multiple compute nodes. DRAW can succeed the two goals: maximize the parallel distribution of the grouping data, and balance the complete storage loads. An observation of the MapReduce framework is that the framework generates a large amount of intermediate data. Such abundant information is thrown away after the tasks finish, because MapReduce is unable to utilize them. So we implement Dache in Hadoop by extending relevant components. Our system eliminates all the duplicate tasks in MapReduce jobs and it is responsible for the performance enhancement of the system. In the future, we plan to adapt our framework to more general application scenarios and implement the scheme in the Hadoop project.

References

[1] Jun Wang, Qiangju Xiao, Jiangling Yin, and Pengju Shang, ”DRAW: A New Data-gRouping-AWare Data Placement Scheme for Data Inten-sive Applications With Interest Locality”, IEEE TRANSACTIONS ON MAGNETICS, VOL. 49, NO. 6, JUNE 2013.

[2] Rongxing Lu, Hui Zhu, Ximeng Liu, Joseph K. Liu, and Jun Shao, ”Toward Efficient and Privacy-Preserving Computing in Big Data Era”, IEEE Network July/August 2014.

[3] Qi Chen, Cheng Liu, and Zhen Xiao, ”Improving MapReduce Perfor-mance Using Smart Speculative Execution Strategy”, IEEE TRANSAC-TIONS ON COMPUTERS, VOL. 63, NO. 4, APRIL 2014.

[4] Shanjiang Tang, Bu-Sung Lee, Bingsheng He,”DynamicMR: A Dy-namic Slot Allocation Optimization Framework for MapReduce Clus-ters”, DOI 10.1109/TCC.2014.2329299, IEEE Transactions on Cloud Computing,2168-7161 (c) 2013 .

[5] Daniel E. Oleary, ”Artificial Intelligence and Big Data”, 1541-1672/13/$31.00 2013 IEEE, iEEE iNTElliGENT SYSTEMS Published by the IEEE Computer Society.

[6] N. Monica, K. Ramesh Kumar, ”Survey on Big Data by Coordinating Mapreduce to Integrate Variety of Data”, International Journal of Science and Research (IJSR) ISSN (Online): 2319-7064

[7] Jeffrey Dean, Sanjay Ghemawat, ”MapReduce: Simplified Data Process-ing on Large Clusters”, unpublished.

[8] Kudakwashe Zvarevashe, Mainford Mutandavari, Trust Gotora, ”A Sur-vey of the Security Use Cases in Big Data”, International Journal of Innovative Research in Computer and Communication Engineering(An ISO 3297: 2007 Certified Organization)Vol. 2, Issue 5, May 2014

[9] D.Usha , Aslin Jenil A.P.S, ”A Survey of Big Data Processing in Perspective of Hadoop and Mapreduce”, International Journal of Current Engineering and Technology E-ISSN 2277 - 4106, P-ISSN 2347 - 5161

[10] X. Jiong, Y. Shu, R. Xiaojun, D. Zhiyang, T. Yun, J. Majors, A. Man-zanares, and Q. Xiao, ”Improving mapreduce performance through data placement in heterogeneous hadoop clusters”, Apr. 2010

[11] B.Thirumala Rao, Dr. L.S.S.Reddy, ”Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments”, International Journal of Computer Applications (0975 - 8887) Volume 34- No.9, November 2011 29

[12] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, ”MapReduce Online”, unpublished.

[13] Nimrod Megiddo and Dharmendra S. Modha, “ARC: A SELF-TUNING, LOW OVERHEAD REPLACEMENT CACHE”, USENIX Association, 2nd USENIX Conference on File and Storage Technologies.

[14] Hossein Rahimi, Fereshteh-Azadi Parand , Davoud Riahi, "Hierarchical simultaneous vertical fragmentation

and allocation using modifieded Bond Energy Algorithm in distributed databases", Saudi Computer Society, King Saud University, Applied Computing and Informatics

Author Profile

Sushant Nagavkar received the B.E. degree in Information Technology with first class from DKTE college of Engineering and textile Ichalkaranji under Shivaji University Kolhapur in 2013. Now I am with GHRCEM college of Engineering and Management, Ahemadnagar, Maharashtra under Savitribai Phule Pune University, appearing M.E. degree in Computer Networks.